ALGORITHMS FOR DETERMINING AND LABELLING APPROXIMATE HIERARCHICAL SELF-SIMILARITY

Christophe Rhodes, Michael Casey
Department of Computing
Goldsmiths, University of London
London SE14 6NW

ABSTRACT

We describe an algorithm for finding approximate sequence similarity at all scales of interest, being explicit about our modelling assumptions and the parameters of the algorithm. We further present an algorithm for producing section labels based on the sequence similarity, and compare these labels on some expert-provided ground truth for a particular set of recordings.

1 INTRODUCTION

Methods for detecting similar regions in music recordings have many applications, for example in music summarization; song identification; audio compression; and content-based music query systems. Approaches to similarity detection and segmentation of musical audio have been based on many audio features, such as timbre or 'the way it sounds' [1, 2], chroma or harmonic features [3], or partial transcription [4].

We present in this paper a top-down method for generating a tree of regions within a track related by similarity, where that similarity is defined by the user's choice of audio feature and processing method, by an acceptable error rate, and by the predicate for determining whether two sequences match; we further present a method for assigning linear structure labels to regions given such a tree. We discuss our motivation in section 1.1 and related work in section 1.2, before presenting our algorithms in section 2. Some preliminary experimental validation is presented in section 3, and we conclude in section 4.

1.1 Motivation

The initial motivation for this work was provided by the CHARM [1] project, with an inquiry about finding similar regions in audio tracks, with particular reference to almost-literal repeats in recordings of Chopin Mazurkas. In fact, with a known score, the approach to solving that task would likely be very different from use of the algorithms presented here: an approach based on aligning the score (or a MIDI version) to the recoded audio [5, 6], and looking for discontinuities (which would indicate a repeat or an omitted section).

However, in more general contexts, it is important to be able to identify repeated sections with less *a priori* knowledge than having a notated score with written-out repeated sections: this paper considers primarily working directly from recorded audio, though the techniques described are applicable to finding structure in music in transcribed formats (such as MIDI).

A secondary motivation behind the approach that we took is to minimize the number of parameters in the algorithm, and to have those parameters which remain have a straightforward interpretation in terms of the original sequence being investigated for self-similarity.

We first aim to identify pairs of regions which match each other (with a certain allowed error rate). We make some assumptions about the structure of the matches that we are interested in, the primary assumption being that the matched regions are arranged in a hierarchical fashion: that boundaries on large scales are not crossed by smaller-scale matches. Note that we do not wish to claim that all musical structures are arranged in a single hierarchy, but that when working with one particular kind of structure (induced over one particular audio feature) it is likely that a hierarchical arrangement is a reasonable approximation.

Once we have identified the regions related by pairwise similarity, we also wish to summarize this information in some simple way; in order to compare the results from our algorithm with ground truth from the CHARM project, we derive structure labels from the pairwise similarity data.

Note that a simple application of sequence alignment as commonly used in bioinformatics [7] is not appropriate for this problem, as our hierarchical criterion leads us to prefer long acceptable matches over short 'better' ones (see figure 1).

1.2 Related Work

Many existing methods of determining areas of track self-similarity are based on the S-matrix [8] containing a measure of dissimilarity for short-time feature vectors; a three-minute song produces a matrix of 18000×18000 entries. This large object, related to recurrence plots [9], is then investigated for diagonal lines of low dissimilarity. Equiv-

[1] Centre for the History and Analysis of Recorded Music.

a	b	c	d	e	f	g	h	i	j
a	b	k	d	e	l	m	h	i	n
0	0	1	0	0	1	1	0	0	1
1	2	0	1	2	0	-2	-1	0	-2

Figure 1. The upper two rows contain two substrings of length ten being considered for matching, with the per-character match summarized in the third row (0 for matching, 1 for a mismatch at a given position). The lowest row shows the alignment for a match score of 1 and a substitution penalty of 2. A sequence alignment would prefer a match of length 2 or 5, whereas given an error rate of $\frac{1}{3}$ we wish to consider the first nine characters as our preferred match.

alently, the time-lag matrix as used in [10, 11] and [12] is a rotation of half of the S-matrix, and regions related by similarity are indicated by horizontal lines. These authors then post-process their matrices (in whatever orientation) by operations inspired by image processing (*e.g.* erosion and dilatation), to attempt to enhance the relevant regions and eliminate noise; then lags corresponding to repeated segments are detected by averaging the dissimilarity for a given lag and thresholding. Where these previous works use short-time audio features and simple smoothing techniques, others (such as [13]) generate a smoothed S-matrix by applying dynamic time warping to regions intermediate in size between individual audio frames and likely segment sizes.

The problem of assigning structure labels to tracks is addressed at least implicitly (and sometimes explicitly) in some of these works. In some, the task at hand was to detect specific kinds of segment (the chorus in [10, 11], for example), and so only that subtask was addressed, though some treatment of transitive closure of pairwise relations is discussed. In [12], heuristic methods for converting from pairwise-similar regions to structure labels are discussed, along with methods for dealing with overlaps; the method of [14] for building an 'explanation' of pairwise or clustered structure resembles the structure labels that we generate, though the explanation is sensitive to the order in which the pairwise clusters are processed. There is a discussion of structure labelling and tree similarity from a bottom-up viewpoint in [13]; the labelling suffers from overlap conflicts which are resolved by an ordering by repetition count in a potentially lossy way, in contrast to the scheme described in section 2.2 below.

2 ALGORITHMS

We take as given a string S of symbols over a given alphabet of length L, and a matchp predicate, which evaluates whether two substrings of a given length (from the same alphabet) match. The matching is such that a certain per-character error rate $0 \leq \alpha \leq 1$ is acceptable, and that per-character error rate is a constant for all substring lengths: this formulation of matching allows a form of memoization, in that if the number of mismatches between the start of the substrings and position $p < l$ exceeds αl, the maximum permitted errors for a match of length l, then the maximum number of permitted errors $\alpha l'$ for matches of length l' for $p \leq l' < l$ will also be exceeded by at most position p (see figure 2 for an illustration of this).

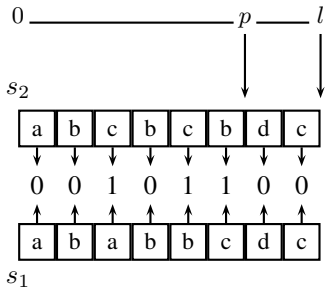

Figure 2. An illustration of matchp's behaviour. In this example, for a prospective match of length $l = 8$, with $\alpha = \frac{1}{4}$, we find after checking the sixth character that we have exceeded the maximum allowed number of errors (2) for matches of length 8, and so that we need not check this pair of start points until l is smaller than 6 (and eventually a match will be found for these start positions at $l = 4$.

We also assume a substr subroutine which extracts or otherwise indicates a substring of a string given a start point and a length.

Although we have phrased the algorithm in terms of a string over a finite alphabet of symbols, it is straightforward to adapt it to a vector of continuous observations of arbitrary dimensions in a finite metric space, with matchp adapted to consider a normalized distance measure between observations instead of a boolean comparison between symbols.

2.1 Generating Pairwise Matches

The first piece of our overall algorithm is the nextPossiblePair function described in algorithm 1, which finds the next possible pair of start indices s_1, s_2 (given the current values) for a match of length l in a string of total length L.

Algorithm 1 nextPossiblePair$(s_1, s_2, l, L) \rightarrow s_1', s_2'$

```
if s_1 = L − 2l then
    return ⊥
else if s_2 = L − l then
    return s_1 + 1, s_1 + 1 + l
else
    return s_1, s_2 + 1
end if
```

Algorithm 2 is a brute-force method for finding the longest matching matching regions in a string; it is too slow for our purposes: for length l matches in a string of length L, there are

$$\sum_{i=0}^{L-2l} (L - 2l + 1 - i) = \frac{1}{2} (L - 2l + 1) (L - 2l + 2)$$

possible start pairs, each of which will do work Cl to perform the matchp operation. In the worst (no match) case, we do this for all $l_0 \leq l \leq \frac{L}{2}$, giving overall work of $O(L^4)$.

Algorithm 2 Longest pairwise match, brute force

for l downfrom $\lfloor \frac{L}{2} \rfloor$ to l_0 **do**
  $(s_1, s_2) \leftarrow (0, l)$
  repeat
    if matchp(substr(S,s_1,l),substr(S,s_2,l)) **then**
      return s_1,s_2,l
    end if
    $(s_1, s_2) \leftarrow$ nextPossiblePair(s_1, s_2, l, L)
  until $(s_1, s_2) = \perp$
end for

Even if the constant terms in front of the highest-order terms are small, this is prohibitively expensive for strings corresponding to audio tracks at, say, one symbol per second. We can, however, improve on this with relatively little effort, making this search practical for the sizes of strings that we are dealing with. We can build a cache A_{ij}, indexed by start positions i, j, of positions p at which the matchp predicate discovered that the per-character error rate for a match of length l would be greater than the permitted error rate. This then implies that the per-character error rate for any smaller match $l' \geq p$ must also be larger, so we do not need to call matchp again with those start indices until the length of the putative match is less than p.

Algorithm 3 Longest pairwise match, cacheing

$A_{ij} \leftarrow \lfloor \frac{L}{2} \rfloor + 1$ for all $0 \leq i, j < L$
for l downfrom $\lfloor \frac{L}{2} \rfloor$ to l_0 **do**
  $(s_1, s_2) \leftarrow (0, l)$
  repeat
    if $l < A_{s_1 s_2}$ **then**
      $(m, p) \leftarrow$ matchp(substr(S,s_1,l),substr(S,s_2,l))
    end if
    if m **then**
      return s_1,s_2,l
    else
      $A_{s_1 s_2} \leftarrow p$
    end if
    $(s_1, s_2) \leftarrow$ nextPossiblePair(s_1, s_2, l, L)
  until $(s_1, s_2) = \perp$
end for

We thus amortise the work of matchp, Cl, over $l - p \sim (1 - \alpha)l$ comparisons (where α is the allowed error rate), thus reducing the overall complexity of the algorithm to $O(L^3)$ at a cost of $O(L^2)$ space; note that the smaller α is, the lower the constant of proportionality in front of the L^3.

At no extra cost in work, we can turn this into an algorithm for finding all relevant matches at all length scales of interest is a matter of tracking two more pieces of information: the matches themselves, and the inferred boundaries: when a match is found, the new boundaries alter the generation of all subsequent possible s_1, s_2 pairs.

Algorithm 4 nextPair$(s_1, s_2, l, L, B) \rightarrow s_1', s_2'$

local predicate admissiblePair(s_1, s_2, l, B):
$\nexists i : [(s_1 < i < s_1 + l) \vee (s_2 < i < s_2 + l)] \wedge (i \in B)$

$s_1, s_2 \leftarrow$ nextPossiblePair(s_1, s_2, l, L)
if $(s_1, s_2) = \perp$ **then**
  return $\perp$
else if admissiblePair(s_1, s_2, l, B) **then**
  return s_1, s_2
else
  return nextPair(s_1, s_2, l, L, B)
end if

In algorithm 4, the admissiblePair local predicate determines whether the proposed pair of regions (designated by start indices s_1,s_2 and length l) overlaps any already-detected boundaries (in B). One simple way of implementing this simply is to represent the string S as a linked list of regions between boundaries, performing list splicing in constant time when new boundaries are identified. Algorithm 5 additionally ensures that once two regions have been identified as being pairwise related, then no pairs of subregions from those regions will be considered. This will not prevent us from finding relevant substructure, however, as any such will necessarily have at least one pair of regions not so excluded.

Algorithm 5 All pairwise matches

$A_{ij} \leftarrow \lfloor \frac{L}{2} \rfloor + 1$ for all $0 \leq i, j < L$
$B \leftarrow \{\}$; $M \leftarrow \{\}$
for l downfrom $\lfloor \frac{L}{2} \rfloor$ to l_0 **do**
  $(s_1, s_2) \leftarrow (0, l)$
  repeat
    if $l < A_{s_1 s_2}$ **then**
      $(m, p) \leftarrow$ matchp(substr(S,s_1,l),substr(S,s_2,l))
    end if
    if m **then**
      $A_{ij} \leftarrow 0$ for $s_1 \leq i < s_1 + l, s_2 \leq j < s_2 + l$
      $B \leftarrow B \cup \{s_1, s_1 + l, s_2, s_2 + l\}$
      $M \leftarrow M \cup \{(s_1, s_2, l)\}$
    else
      $A_{s_1 s_2} \leftarrow p$
    end if
    $(s_1, s_2) \leftarrow$ nextPair(s_1, s_2, l, L, B)
  until $(s_1, s_2) = \perp$
end for
return M

2.2 Assigning Labels

The algorithm in section 2.1 generates a set of pairwise-matched regions. This contains all the information that is needed; however, structure labels are a good way of summarizing this information (see *e.g.* [13]).

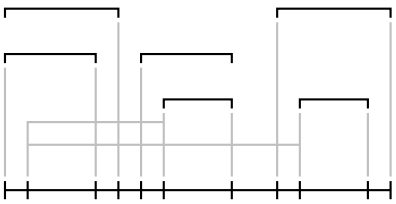

Figure 3. Illustration of transitive closure: the top half of the diagram represents detected pairwise matches, while the line at the bottom is the division into regions. Note the leftmost small region, which is induced by the pairwise similarity of a region which to another which itself contains a match.

As a first step towards producing a summary, we will divide up the sequence into regions whose points share the same symmetries, those symmetries being the transitive closure of the pairwise similarities (see figure 3). Then, to generate structure labels, we will assign labels to regions in decreasing order of size until each pairwise match from the original detection contains at least one label (and until all unlabelled regions are sufficiently small). This has an effect similar to the heuristics given in the structure analysis section of [12] and the cluster splitting in [14].

Algorithm 6 Transitive closure from pairwise matches M

$T_i \leftarrow \{\}$ for all $0 \leq i < L$
for (s_1, s_2, l) in M ordered by ascending size **do**
  $T_{s_1:s_1+l} \leftarrow T_{s_1:s_1+l} \cup T_{s_2:s_2+l} \cup \{(s_1, s_2, l)\}$
  $T_{s_2:s_2+l} \leftarrow T_{s_1:s_1+l}$
end for
return T

Algorithm 6 illustrates computation of the transitive closure of pairwise matches; because of our hierarchical constraint of these pairwise matches, we can simply iterate over all matches in order of increasing size, as we know that no part of a larger match can be contained in a smaller match. The vector T is then segmented into regions of related similarity, where a region is defined as a contiguous set of entries where the set of transformations is the same and none of those transformations has a boundary in that region. This segmentation by transformations contains the equivalent information to the pairwise similarities, but is in a form that is easier to interpret.

We then label this segmentation by sorting by size of segment (resolving ties by grouping related segments together), and assigning labels in decreasing segment size, continuing until both every pairwise match detected has had at least one label assigned to a subregion, and until the region size is under some salient length l'_0 (which can but need not be the same as l_0 in section 2.1.

3 EXPERIMENTAL DETAILS

Our test corpus consists of twenty-seven recordings of the mazurka in A minor, Op. 7 Nr 2 by Chopin. Table 1 illustrates the structure of the mazurka on various levels: the notated score is in four sections, labelled $\mathcal{A}$, $\mathcal{B}$, $\mathcal{C}$ and $\mathcal{D}$. Sections $\mathcal{A}$, $\mathcal{B}$ and $\mathcal{D}$ are sixteen bars long, and are notated to be repeated; section $\mathcal{C}$ is eight bars long and is only played once. Additionally, the repeats of $\mathcal{B}$ and $\mathcal{D}$ have small differences in the final bar, and a *da capo al fine* is specified, so section $\mathcal{A}$ is notated to be played again (once) at the end.

We convert the audio recordings into a sequence of symbolic labels by performing an initial segmentation by timbral features into five segment classes according to the method of [15], and generating a string with one segment label per second. This segmentation can be an accurate structural segmentation in itself for certain kinds of music [16] but in the case of solo piano music, where timbral changes do not indicate structural changes directly, the effect of this prior segmentation is to perform temporal smoothing of the audio features, allowing a lower value of α and allowing us not to have to perform dynamic time warping. It is important to note that this preprocessing step is independent of the algorithms described herein, which can be used on any sequential data with a normalized distance measure.

Table 2 presents some experimental results, where we used a threshold error rate of $\alpha = \frac{1}{12}$ and a minimum length $l_0 = l'_0 = 10$ corresponding to a time of 10s. Our algorithm working on the processed audio as described above gives labellings corresponding with the (corrected) ground truth in nine of the twenty-seven cases, where we treat our 'CCD' sequence as equivalent to the ground truth 'CDD' for reasons discussed below.

The first thing to note is that there is more structure to this Mazurka than is evident from the 'ground truth' labelling: the first row in table 1 describes the similarity relationships on an eight-bar metrical grid. This substructure explains why we have accepted 'CCD' from our algorithm as equivalent to 'CDD' in the ground truth, as it corresponds to the **deded** section in the actual score: and there is no way of distinguishing from just the audio that it is notated in 'CDD' fashion. Further, we see some of this eight-bar substructure being detected by the algorithm in the recordings by Smith (1975) and Indjic (2001); indeed, the algorithmic answers for those two recordings are a fair reflection of the performance in question.

There are other classes of discrepancy between the algorithmic labels and the ground truth: in three cases, the algorithm has failed to label the 'orphan' segment in the **deded** section (and in some others, there is another single missing segment); in several cases, there is an unmatched label at the end of the string, presumably corresponding to silence. Because of the way our algorithm is structured, the single label for the inner sections of the François recordings (without repeats) is as correct as it can be.

Finally, we note that in the light of recent revelations

a	b	a	b	c	b	c	b	d	e	d	e	d	a	b
A		A		B		B		C	D		D		A	
$\|: \mathcal{A} :\|$				$\|: \mathcal{B} :\|$				$\mathcal{C}$	$\|: \mathcal{D} :\|$				*d.c.*	

Table 1. The structure of Chopin's mazurka in A minor, Op. 7 Nr 2. The top line corresponds to eight-bar units, allowing for small differences in the musical material at the beginning and end of the eight bars. The middle line corresponds to the ground truth labels provided by an expert for a performance corresponding to the notated score represented in the bottom line.

Recording	Algorithmic labels	Ground Truth	(a)	(b)	(c)	(d)
Ashkenazy (1981)	ABCCCDDA	AABBCDDA	×			
Biret (1990)	ABCCDDEB	AABBCDDA	×			
Block (1995)	AABBCDCE	AABBCDDA	×			
Brailowsky (1960)	AABBCCDA	AABBCDDA				
Chiu (1999)	ABCCDDBE	AABBCDDA	×			×
Clidat (1994)	AABBCCA	AABBCDDA			×	
Cortot (1951)	ABCCDEEFAG	AABBCDDA	×			×
Falvay (1989)	ABCCDEEFG	AABBCDDA	×			
Fiorentino (1962)	AABBCCDA	AABBCDDA				
Flière (1977)	AABBCCDA	AABBCDDA				
François(1956)	ABA	ABCDA				
François (1966)	AABA	ABCDA		×		
Friedman (1930)	ABBCCD	AABBCDDA*			×	
Hatto (1997)	ABBCDDA	AABBCDDA*				
Indjic (2001)	ABCBCBDDEB	AABBCDDA*		×		×
Kapell (1951)	AABBCCDA	AABBCDDA				
Luisada (1990)	AABBCCA	AABBCDDA			×	
Magaloff (1977)	AABBCCDA	AABBCDDA				
Pobłolcka (1999)	AABBCCDAB	AABBCDDAB				
Rubinstein (1939)	AABCCDA	AABCDDA				
Rubinstein (1952)	AABBCCDA	AABBCDDA				
Rubinstein (1966)	ABCCDEEFA	AABBCDDA	×			
Shebanova (2002)	AABBCCA	AABBCDDA			×	
Smith (1975)	ABABCBCBDDEABF	AABBCDDA		×		×
Ts'ong (1993)	ABCCDDEA	AABBCDDA	×			
Ts'ong (2005)	AABCCAD	AABBCDDA			×	×
Uninsky (1959)	ABCBCDDCE	AABBCDDA*		×		×

Table 2. The labels derived using the algorithms presented in section 2 for the set of 27 performances of Chopin's Mazurka Op. 7 Nr 2 in A minor. Note that four of the ground truth labels provided by an expert listener (starred) are incorrect, and should read ABBCDDA. Various classes of discrepancy between ground truth and algorithmic labels are summarized on the right of the table: (d) indicates labelling silence at the end of the track; (c) indicates missing one segment; (b) is marked if the algorithm has labelled structure at a finer detail than the ground truth, and (a) is for other errors, most often from failure to detect a pairwise match.

about the discography of Hatto [17], we can assess from these results a certain amount about the sensitivity of the algorithm to the audio processing chain used in this case, as the input audio of the Hatto (1997) and Indjic (2001) recordings is for all practical purposes identical, while the audio processing has some random elements. The difference in algorithmic labels between those two recordings thus indicates that our method as presented is sensitive to features of the audio processing chain.

4 CONCLUSIONS

We have presented methods for detecting and labelling hierarchical structure in sequential data, with a small set of parameters which are straightforwardly interpretable; the preliminary results from these methods on a stringent test are encouraging. The method as presented assumes a matchp predicate which works character-by-character; however, real music often undergoes temporal alterations between regions of similar material. In this investigation, we dealt with that issue by having an elaborate processing chain; however, there is nothing to stop a different matchp predicate including some dynamic time warping: the challenge would be to preserve efficiency. A more straightforward refinement to the method presenting here would be simply to prohibit pairwise matches from being detected as starting or ending on a mismatched character, which could be incoroporated into the initialization of A_{ij}. Finally, we note again that the methods presented here are not specific to audio processing, and are in use in analysis of a large database of MIDI performance transcriptions.

Acknowledgments

We thank Craig Sapp and Raphael Clifford for helpful discussions. This work was supported by EPSRC grant GR/S84750/01.

5 REFERENCES

[1] Jean-Julien Aucouturier, Françcois Pachet, and Mark Sandler. The Way It Sounds: Timbre Models For Analysis and Retrieval of Polyphonic Music Signals. *IEEE Transactions of Multimedia*, 2005.

[2] Samer Abdallah, Katy Noland, Mark Sandler, Michael Casey, and Christophe Rhodes. Theory and evaluation of a Bayesian music structure extractor. In Joshua D. Reiss and Geraint A. Wiggins, editors, *Proc. ISMIR*, pages 420–425, 2005.

[3] Mark A. Bartsch and Gregory H. Wakefield. To Catch a Chorus: Using Chroma-Based Representations for Audio Thumbnailing. In *Proc. WASPAA*, 2001.

[4] N. Maddage, X. Changsheng, M. Kankanhalli, and X. Shao. Content-based Music Structure Analysis with Applications to Music Semantics Understanding. In *6th ACM SIGMM MIR Workshop*, October 2004.

[5] Ferréol Soulez, Xavier Rodet, and Diemo Schwarz. Improving Polyphonic and Poly-Instrumental Music to Score Alignment. In *Proc. ISMIR*, pages 143–148, 2003.

[6] Christopher Raphael. A Hybrid Graphical Model for Aligning Polyphonic Audio with Musical Scores. In *Proc. ISMIR*, pages 387–394, 2004.

[7] M. S. Waterman and M. Eggert. A New Algorithm for Best Subsequence Alignments with Application to tRNA-rRNA Comparisons. *Journal of Molecular Biology*, 197:723–728, 1987.

[8] Jonathan Foote. Visualizing Music and Audio using Self-Similarity. In *ACM Multimedia (1)*, pages 77–80, 1999.

[9] J.-P. Eckmann, S. O. Kamphorst, and D. Ruelle. Recurrence Plots of Dynamical Systems. *Europhysics Letters*, 5:973–977, 1987.

[10] Masataka Goto. A chorus-section detecting method for musical audio signals. In *Proc. ICASSP*, volume V, pages 437–440, 2003.

[11] Matasaka Goto. A Chorus Section Detection Method for Musical Audio Signals and Its Application to a Music Listening Station. *IEEE Transactions on Audio, Speech and Language Processing*, 14(5):1783–1794, 2006.

[12] L. Lu, M. Wang, and H. Zhang. Repeating pattern discovery and structure analysis from acoustic music data. In *6th ACM SIGMM MIR Workshop*, October 2004.

[13] Wei Chai. *Automated Analysis of Musical Structure*. PhD thesis, MIT, 2005.

[14] R. Dannenberg and N. Hu. Discovering musical structure in audio recordings. In *Music and Artifical Intelligence: Second International Conference*, Edinburgh, 2002.

[15] Samer Abdallah, Mark Sandler, Christophe Rhodes, and Michael Casey. Using duration models to reduce fragmentation in audio segmentation. *Machine Learning*, 62(2-3):485–515, 2006.

[16] Christophe Rhodes, Michael Casey, Samer Abdallah, and Mark Sandler. A Markov-Chain Monte Carlo Approach to Musical Audio Segmentation. In *Proc. ICASSP*, volume V, pages 797–800, 2006.

[17] Nicholas Cook and Craig Sapp. Purely coincidental? Joyce Hatto and Chopin's Mazurkas. `http://www.charm.rhul.ac.uk/content/contact/hatto_article.html`.